Degani A. and Heymann M. (2002) Formal verification of human-automation interaction. *Human Factors,* 44(1), pp 28-43.

FORMAL VERIFICATION OF HUMAN-AUTOMATION INTERACTION

ASAF DEGANI
NASA Ames Research Center, Mountain View, California
adegani@mail.arc.nasa.gov

MICHAEL HEYMANN
Department of Computer Science
Technion, Israel Institute of Technology
heymann@cs.technion.ac.il

ABSTRACT

This paper discusses a formal and rigorous approach to the analysis of operator interaction with machines. It addresses the acute problem of detecting design errors in human-machine interaction and focuses on verifying the correctness of the interaction in complex and automated control systems. The paper describes a systematic methodology for evaluating whether the interface provides the necessary information about the machine, so as to enable the operator to perform a specified task successfully and unambiguously. It also addresses the adequacy of the information, provided to the user via training material (e.g., user manual), about the machine's behavior. The essentials of the methodology, which can be automated and applied to the verification of large systems, are illustrated by several examples and through a case study of pilot's interaction with an autopilot onboard a modern commercial aircraft.

Running head: human-automation interaction.

Key words: automation, modeling, design of interfaces, formal-methods, verification, cockpit design.

INTRODUCTION

With the accelerated introduction of advanced automation into a variety of complex, human-operated systems, unexpected problems with overall system performance have been observed (Parasuraman, Sheridan, and Wickens, 2000; Wiener, 1989, 1993; Wiener, Chute, and Moses, 1999). Many of these problems have been attributed to deficiencies in communication and coordination between the human and the machine. They are especially acute in cases where the human and the machine share authority over the system's operation (Mellor, 1994). Notable examples of such systems are modern commercial aircraft with advanced flight management systems (Abbott, Slotte, and Stimson, 1996).

One important and well-documented class of advanced automation systems are Automatic Flight Control Systems (AFCS) of modern jetliners. In recent years, faulty pilot interaction with the AFCS has become a major concern in the civil transport industry. This problem has variously been termed as lack of mode awareness, mode confusion, or automation surprises (Woods, Sarter, and Billings, 1997). Two main factors have frequently been cited in accident and incident reports and in the scientific literature, as being responsible for such breakdowns. (1) The user has an inadequate “mental model” of the machine’s behavior (Javaux and De Keyser, 1998; Sarter and Woods, 1995). (2) The interface between the user and the machine provides inadequate information about the status of the machine (Indian Court of Inquiry, 1992; Norman, 1990). Both factors may limit the user's ability to reliably predict or anticipate the next configuration (e.g., mode) of the machine, and hence may lead to false expectations, confusion, and error (Degani, Shafto, and Kirlik, 1999).

Traditionally, most of the human factors research on interface design has focused on perceptual and cognitive compatibility between the human and the interface format (e.g., Wickens, 1992). Much less research was conducted on the relationship between the interface and the machine being

controlled. Notable exceptions are the current work in cognitive and ecological psychology (see for example Vicente, 1999), and the past work in the area of manual control (Rouse, 1977).

Faulty interaction of the user with the machine, which can lead to catastrophic results in high-risk systems such as commercial aircraft, is variously attributed to either machine failures, or human errors, with the latter sometimes blamed on interface design inadequacies. However, the distinction between human error and interface design inadequacies has remained blurred (see e.g. Aviation Week and space Technology, 1995). Furthermore, the possibility of failures because of ambiguous responses of the machine to pilot interaction, has not received much attention to date. The reason for this is the complexity of such advanced automation systems and the absence of rigorous methods for their analysis.

The objective of the present paper is to propose a formal and systematic approach for verifying human-machine systems. In particular, a formal procedure will be presented for deciding whether a given human-machine system is adequate for its task. That is, whether the interface enables the user to interact with the machine correctly and reliably so as to achieve specified operational goals. The proposed procedure can be automated and applied to the verification of large and complex human-machine systems. In the present paper we focus on the heuristic aspects of the approach and demonstrate it via two illustrative examples. The more mathematical aspects of the methodology are discussed in Degani, Heymann, Meyer, and Shafto (2000).

The paper is organized as follows: First, we discuss the role of the task specification as related to reliable human-machine interaction. In particular, we examine the interrelations among the machine's behavior, the user-interface, the user's model of the machine, and the task. We follow with an introductory description, using an example, of how a formal analysis can be performed to verify the adequacy of the user-model and interface for a specified task. This is followed by a description of a

formal methodology for verification of user-model (and interface) adequacy. Finally, the methodology is demonstrated by applying it to the analysis of pilot's interaction with a modern autopilot.

ELEMENTS OF HUMAN-MACHINE INTERFACES

In the interaction between the user and the machine, four elements play central roles. These are (1) The machine's behavior; (2) The operational goals, or task specification; (3) The model that the user has about the machine's behavior (called the *user-model*); and (4) The user interface (through which user obtains information about the machine's state and responses). These four elements must be suitably matched, as we shall demonstrate below, in order to insure correct and reliable user-machine interaction.

Before we examine this issue, we need to make several basic assumptions:

- The underlying machine's behavior is modeled within the framework of a well-defined modeling formalism.
- The machine's behavior is deterministic; that is, at each (internal) state of the machine, its response to every action by the user or to external signals is unique and unambiguous.
- The set of operational requirements (tasks) is specified.
- The user's knowledge of the machine's behavior (obtained e.g. from user-manuals and training) is formally represented.

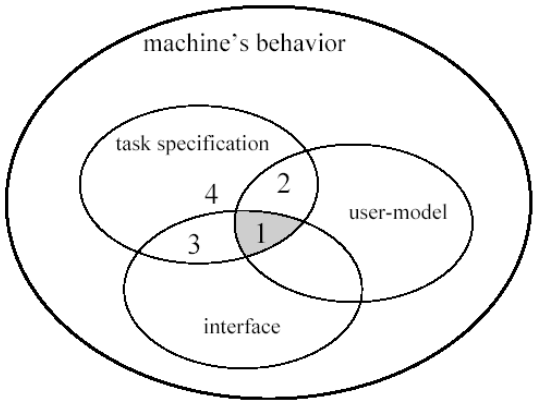

Figure 1. Elements of human-machine interaction

Figure 1 schematically describes the interrelation among these elements. The large circle represents the set of all machine behaviors. Each of the three inner circles represents the region where one of the elements is 'adequate.' In the present paper we shall always assume that the machine's behavior is given and the task is formally specified. We are interested in considering the interrelation among the "task specification," the "user-model," and the "interface". Thus, region 1 represents the situation where the user-model and interface are in correct correspondence with the task specification, and hence correct interaction is possible. Region 2 represents the situation where the user-model is correct for the task specification, but the interface is incorrect. Region 3 represents the situation where the interface is correct for the task specification, but the user-model is not. Finally, region 4 represents the case where both the interface and the user-model are incorrect for the task.

There are a variety of available modeling formalisms that can be employed for representing machine- and user-models. These include state-transition systems (such as automata, Statecharts, object-oriented methods), various rule-based formalisms, Petri-nets, and temporal logics, among others. In this paper we have chosen to employ finite state machines for implementing our approach. This formalism is the most basic and commonly used. It is also well understood and easy to analyze. Since the Finite State Machine theory is the foundation of many current modeling formalisms, the principles of the proposed approach to verification of human-machine interaction are readily adaptable to any of the other methods.

To illustrate the possible discrepancies between the machine's behavior, the user-model, and the interface in relation to reliable task execution, consider the simple machine described in Figure 2(a).

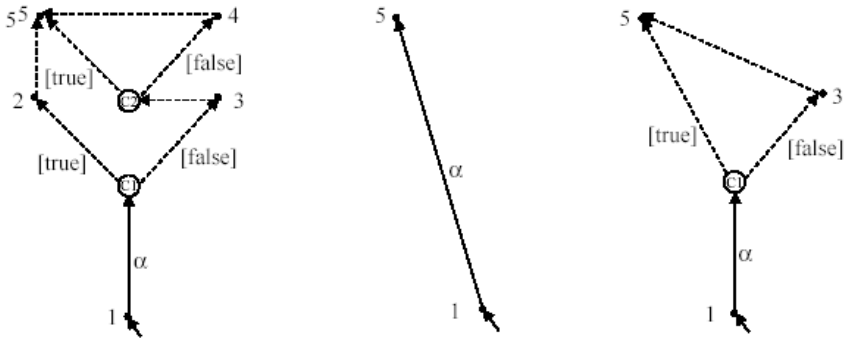

Figure 2(a). Simple machine example

Figure 2(b). User model for specification S1.

Figure 2(c). User model for specification S2.

Figure 2. Example of machine and specifications

The system starts at state 1, and upon execution (by the user) of event "α", moves along to either state 2 or state 3, according to whether the condition "C1" is true or false. The dashed arrows represent transitions that take place automatically and instantaneously without the user's intervention. Thus, if state 2 is reached, the system moves automatically to state 5, while if it reaches state 3, it moves to either state 5 or to state 4 depending on whether condition "C2" is true or false.

Suppose that the task specification (call it S1) is just to drive the system to state 5. In this case, regardless of whether conditions "C1" and/or "C2" are true or false, this will be the guaranteed outcome of executing event "α". The user does not need to know the exact path that the system will take from state 1 to state 5. A reduced (but still correct) user-model for the system of Figure 2(a) and specification S1 is shown in Figure 2(b) -- when at state 1 and event "α" is executed, state 5 will be reached.

On the other hand, if the task specification is to drive the system from state 1 to state 5 via state 3 (call it specification S2), it is necessary that condition "C1" be FALSE. However, since we do not care whether state 4 is visited or not, it does not matter whether condition "C2" is true or false. A correct user-model of the system in this case must exhibit the possible paths implied by condition "C1," but the user-model can still be a reduced description of the system as given in Figure 2(c). To correctly

interact with the system and execute the task specification (S2), the user only needs to know whether condition C1 is true or false before executing "α". To perform the task correctly and reliably, the user must not only have a correct user-model, but also a correct interface. The case in which the user has a correct model (for the specification), but is not provided with a correct interface (e.g., indicating whether condition C1 is true or false) is, as was stated earlier, represented by region 2 of Figure 1. Clearly, in this case, reliable task execution cannot be guaranteed.

Another possibility is that the user has an incorrect user-model. In particular, suppose the user is unaware of the existence of condition C1 (Figure 2(b)), and he or she assumes that the transition to state 5 always traverses state 3. An adequate interface that indicates the status of condition C1 would not be of any help in this instance, since the user would not associate its value with the machine's behavior. (This is the case represented by region 3 of Figure 1.)

FORMAL ASPECTS OF HUMAN-MACHINE INTERACTION

Suppose we are given a new machine, its interface, and a user-model proposed by the engineering design team. As true in most cases, the interface displays an abstracted subset of the actual events and behaviors that occur in the machine. Correspondingly, the user-model is also a simplified description of the machine's behavior. We are asked to verify whether the proposed interface and associated user model are correct for the tasks that the user is expected to perform with this machine.

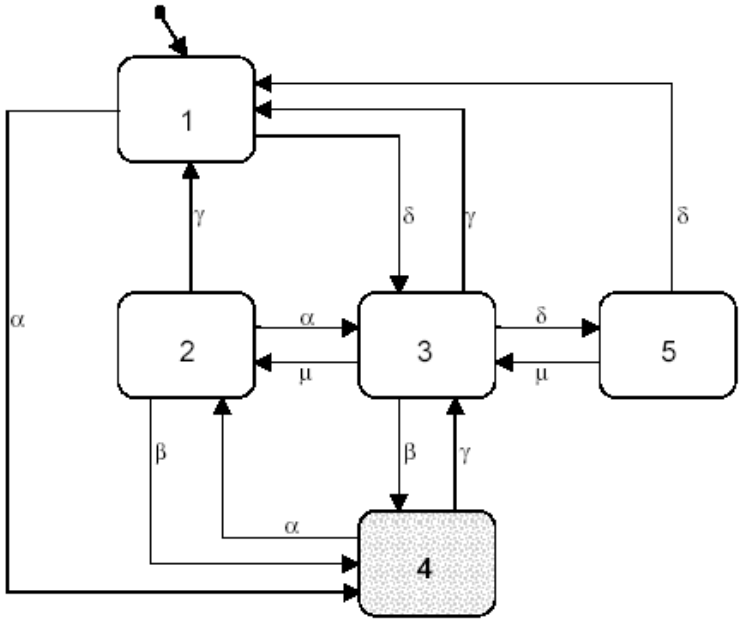

Figure 3: Machine model

Machine Model

Let us consider the machine model described in Figure 3. It consists of 5 states (represented by the numbered boxes) and a set of transitions between these states (represented by labeled arrows). The specification is that states 1,2,3 and 5 are *legal* states, while state 4 (the dotted box) represents an *illegal* state. The user's task is to control the system so as to avoid entering illegal state 4. (The reason for declaring a state to be illegal may be because the state is unsafe or unauthorized.) A proposed user-model for this machine is given in Figure 4. Since the user-model is based on and relates to the events that are displayed in the interface, all the displayed events appear in the user-model. Hence, the interface can be thought of as already "embedded" in the user-model. We therefore perform the analysis on the user-model only.

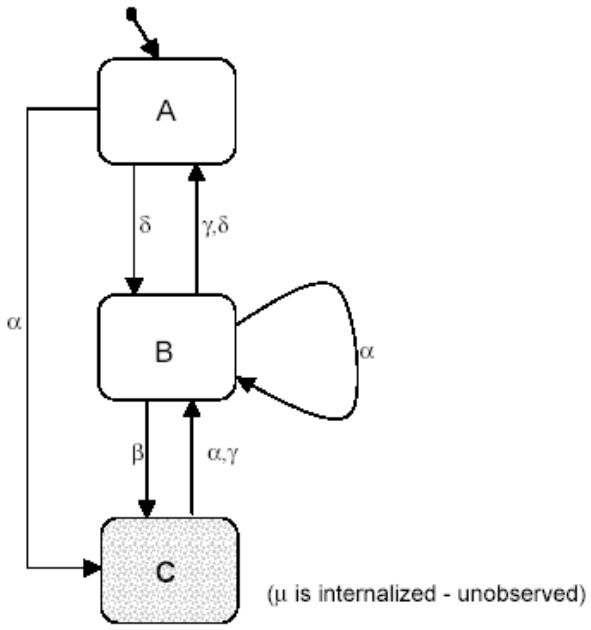

Figure 4. User model

User Model

All the events (transition labels) of the machine appear also in the user-model except for machine event μ that is absent from the user-model (i.e., μ is an unobserved event). In the user-model of Figure 4, states A and B are considered to be the legal states while state C is considered to be illegal. The user interacts with the machine through the interface where he or she observes and triggers transitions. The user decides whether the progress of the machine is legal or not, according to its observed progress on the interface and in the associated user-model. (We remind the reader that the user-model is not explicitly given in the form of a state transition diagram, but presumably can be translated into one.)

Verification Process

Let us try to verify whether the proposed user-model is correct for the task of tracking the machine and making sure that it does not reach an illegal state. The machine of Figure 3 starts at state 1 and the proposed user-model for the machine, Figure 4, starts at state A. The machine can initially respond to event α and move to state 4, or to event δ and move to state 3. The user tracks the progress of the

machine via the user-model, which performs corresponding transitions, in response to the events α and δ, to states C and B, respectively. Thus, when the user interacts with the machine, these two models run in synchrony. This synchronous run can be thought of *figuratively* as the operation of a "composite machine", which for the models of Figures 3 and 4 is presented in Figure 5.

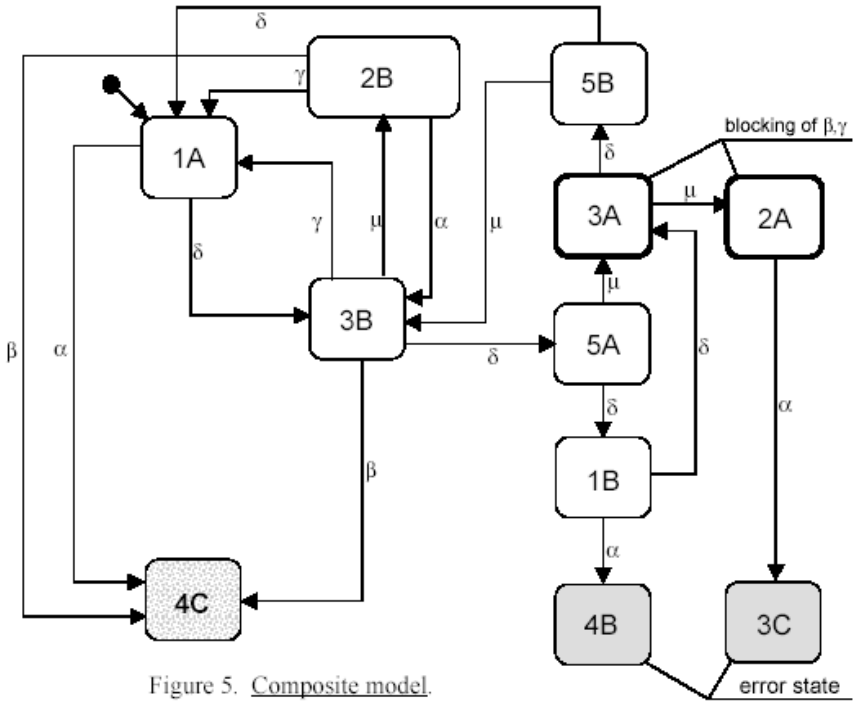

Figure 5. Composite model.

Error states

The basis of the verification procedure rests on the parallel and synchronized execution of the two models. The user's expectation is that when the user-model enters a legal (or illegal) state, the machine also enters a corresponding legal (or illegal) state. Thus, in the composite machine, when a state-pair (that is, a user-model state and a machine state) is entered, either both are legal or both are illegal.

The composite machine of Figure 5 starts in state 1A (i.e., the machine starts in state 1 and the user model in state A). It moves to state 4C in response to event α. Notice that when the machine-model enters state 4, which is illegal, the user-model enters state C that was also termed illegal. This is

consistent with the user's expectation. The composite machine can also move to state 3B in response to event δ. Similarly, the entrance to state 3B is also consistent, since both states 3 and B are legal. If the next event is δ, the composite machine will now move from 3B to 5A, again a consistent transition. Another δ will take it to state 1B and still all is well.

But notice what happens if now, when the machine is in state 1 and the user-model is in state B (i.e., composite state 1B), and event α occurs. The machine moves to the illegal state 4, while the user-model stays in the seemingly legal state B. This is a contradiction -- the user thinks that the move is legal while in reality he or she just drove the machine into an illegal state! We call such a composite state an *error-state*.

This discrepancy cannot be rectified by simply calling the state B illegal because in that case the state pairs 1B, 2B, 3B and 5B would all become error states. Indeed, the set of state pairs 1B, 2B, 3B, 4B and 5B is mixed (i.e., state 4 is illegal while 1, 2, 3 and 5 are legal): There is no resolution of this problem and the user-model must be considered inadequate.

In Figure 5 notice also the error-state 3C, where the user thinks that a move to it is illegal (i.e., leading the user-model to the illegal state C), while in reality, the machine enters the perfectly legal state 3. Again, this could not be remedied by calling the state C legal, because this would make the state-pair 4C an error state. States 3 and 4 are not in the same *specification class*. Obviously, the ability of the composite model to enter such error states implies that the suggested user-model is inadequate and must be rejected. But as we shall see immediately, not only error states must be considered; there is another possibility for inadequate interaction.

Blocking states

Suppose that after entering state 1B (through 3 successive occurrences of δ starting from 1A), event δ takes place again. The machine enters state 3 while the user-model enters state A. The composite state is 3A and until now everything is fine. In state 3, the machine can respond to events

β,δ,γ, and μ. In state A, the user-model implies that there are responses only to events α and δ. The events β and γ exist in the user-model (elsewhere), but according to user-model cannot occur in state A. However, the machine can execute these events in state 3. The user-model is therefore misleading! The user-model of Figure 4 can be viewed as *blocking* the user from interacting correctly with the machine. We call a state such as 3A of the composite-machine a *blocking state*. In a blocking state, the user is unaware of certain events that can take place. Consequently, if any such event is triggered (automatically) by the machine, it will surprise the user. Clearly, a correct user-model must be free of blocking states as well.

We conclude this section with an examination of the adequacy of an alternate user-model for the machine of Figure 3 as shown in Figure 6 (this user-model also has 3 states, just as the earlier candidate).

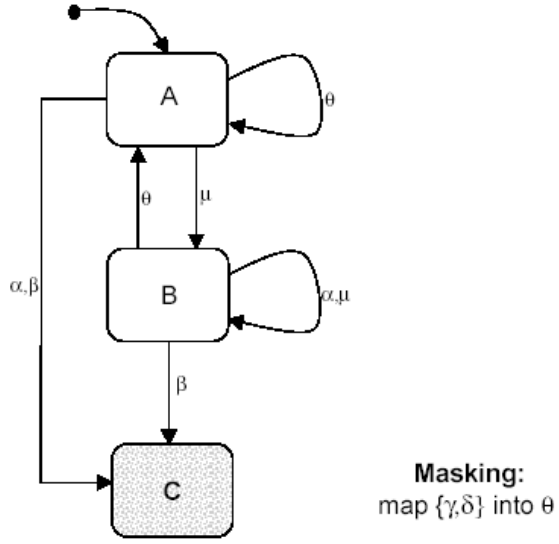

Figure 6. Alternative user model

In this particular user-model μ is not internalized but the events γ and δ of the machine are masked and mapped into θ in the user-model. The composite model is presented in Figure 7 where it can be seen that there are no error and no blocking states. Hence we conclude that this new user-model is correct for the task.

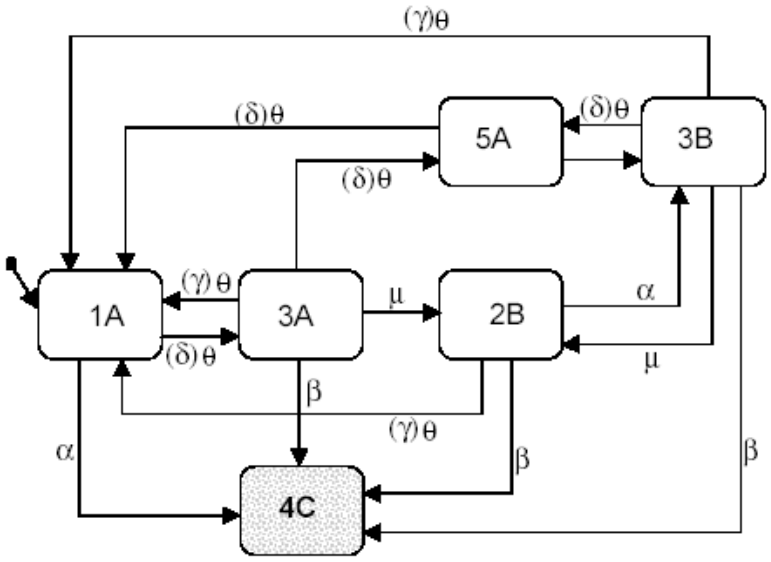

Figure 7. Composite model (with an alternative user model).

FORMAL VERIFICATION OF USER-MODELS

We can summarize our observations about the user-model and display adequacy as follows: The interface and user-model provide an abstracted and reduced description of the machine. This abstracted description does not enable the user to determine with certainty every state the machine is in. However, for adequacy of the interface and user-model, it is only required that the user be able to determine which specification-class the machine is in and which it is about to enter. The verification procedure is aimed precisely at this objective. In addition to this requirement, adequacy of the interface and user-model also requires that the user-model be free of blocking states.

We turn next to a more formal discussion of the verification procedure. Let Σ_M denote the set of events, or *transition labels*, that take place in the actual machine model. The events that appear in the associated user-model, and are displayed in the interface, constitute a reduced subset of the set Σ_M of machine events as explained next.

The event set Σ_M consists of three disjoint subsets: (1) Σ_M^o - the set of *observed-events* that includes all machine events that are actually presented in the interface and appear also in the user-model, (2) Σ_M^m - the set of *masked events* (that are not displayed individually but, rather, are grouped into sets of two or more events each, with each set having a single event-label in the user-model, and (3) Σ_M^u - the set of *unobserved-events* that are neither displayed nor appear in the user-model.

In view of the above, the event set Σ_{USR} of the machine's user-model consists of the union of the event sets $\Pi(\Sigma_M^o)$ (which is identical to Σ_M^o), the event set $\Pi(\Sigma_M^m)$ which denotes the set of events obtained after masking the events in Σ_M^m, and the "empty event" ε (= $\Pi(\Sigma_M^u)$) that represents the set of unobserved events.

In actual operation, the machine is driven by events from Σ_M. The user tracks the progress of the machine via the interface (display), where he or she observes events in Σ_{USR}, with the aid of the associated user-model. Thus, the user-model and the machine evolve concurrently. But they are only partially synchronized in that the user-model tracks the actual state evolution of the machine with some uncertainty. This is because (1) not all machine events are observed and some machine-events are masked, and (2) the user-model is only an abstraction of the actual machine's behavior.

We now need to formally decide whether the interface and associated user-model are in correct correspondence with the specification. That is, we must verify that the tracking uncertainty does not jeopardize the ability of the user to execute the specified tasks reliably. As we have seen in the previous section, the verification consists of showing, that concurrently with every state that the user-model enters, all states that the machine can enter, belong to a single specification class. This implies that there are no error states. Additionally, it must be shown that there are no blocking states. To this end, we proceed by forming the *composite model* obtained by computing the *masked synchronous product* of the machine model and user-model depicted in Figure 8 and explained below.

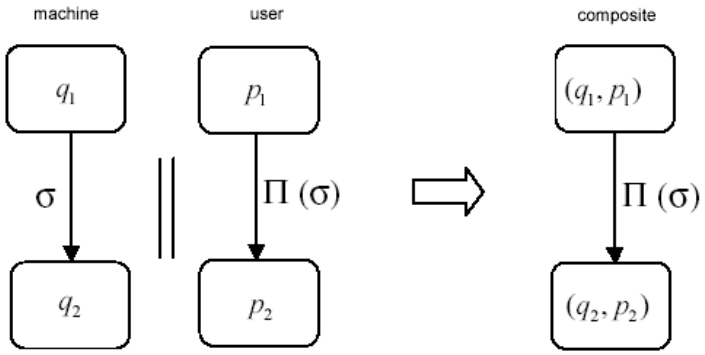

Figure 8. Masked synchronous composition.

This product is formally computed as follows. Suppose that the machine is at configuration *q* at which a transition labeled α is defined, leading to a configuration *q'* (we denote this by $q \xrightarrow{\alpha} q'$). Assume that when the machine is at configuration *q,* the user-model is at a corresponding configuration *p*. Event α, can be either *observed, masked, or unobserved*.

If α is an observed event and hence $\Pi(\alpha) = \alpha$, it is required, for adequacy of the user-model, that a corresponding transition be also defined at configuration *p*, leading to, say, *p'*. That is, there must exist a transition $p \xrightarrow{\alpha} p'$. (Otherwise we would conclude that the user-model exhibits blocking states and is an incorrect abstraction of the machine behavior, regardless of the task specification.) In the composite model there will appear a transition labeled α, from the configuration (pair) (*q,p*) to (*q',p'*). That is, there will be in the composite model a transition $(q, p) \xrightarrow{\alpha} (q', p')$.

If α is a masked event, there must be a corresponding transition $p \xrightarrow{\Pi(\alpha)} p'$ in the user-model, where $\Pi(\alpha)$ is the (masked) image of α in Σ_{USR} (otherwise we would conclude that the user-model is blocking). The transition in the composite model will appear as $(q, p) \xrightarrow{\Pi(\alpha)} (q', p')$. The fact that the event labels in the composite model are taken as those from the user-model is because the composite model is formed "from the point of view of the user".

Finally, if α is unobserved and $\Pi(\alpha) = \varepsilon$, the transition in the composite model will appear as $(q, p) \xrightarrow{\varepsilon} (q', p)$, since there is no corresponding transition in the user-model and the transition is "viewed" by the user as the empty or *silent* transition.

For the user-model to be correct for the task specification, two conditions must be met. First, as stated earlier, the user-model must not block transitions in the composite model. That is, if a transition is defined in the machine model, there must be a *corresponding* transition in the user-model. In case of an *observed* transition there must be a corresponding transition with the same label in the user-model, and in case of a *masked* transition there must be a corresponding transition labeled by the masking image. Thus, in particular, every machine state that is reachable in the machine model is also a component of a reachable state-pair in the composite model, and every transition that is possible in the machine model has a corresponding transition in the composite model. The second condition is that the composite model does not exhibit error states with respect to the task specification.

We can summarize our discussion about user-model and interface verification as follows.

1. Define the event correspondence between the machine-model events and user-model events as discussed above.
2. Form the masked composition between the machine-model and the user-model.
3. Verify that there are no reachable (from the initial state) error states and no reachable blocking states in the composite model.

Naturally, for larger systems with many modes and transitions, the procedure described above can be mechanized and suitable software tools developed. Furthermore, the composite-machine that was constructed here primarily for illustrative purposes need not be constructed explicitly. Rather, a composite "reachability analysis" is performed in testing for error and blocking states.

A CASE STUDY: AUTOMATIC FLIGHT CONTROL SYSTEM

In this section we shall demonstrate the usefulness of the methodology by verifying one element of a flight control system onboard a modern jetliner. For brevity, we shall only investigate a fragment of the transition behavior of the autopilot among several vertical flight modes, with emphasis on the possible pilot's interaction with these mode transitions. The data were obtained through a series of extensive flight simulations using this specific autopilot (Degani et al., 2000). The selected fragment is rather simple, and for this example the analysis can be performed manually. Clearly, for verification of a larger system, a mechanized procedure is required.

We begin by describing the control panel through which the pilot interacts with the autopilot, and the display through which the pilot obtains information about the system's behavior.

Interface Description

Figure 9(a) and 9(b) are schematic illustrations of the relevant elements of the "Guidance Control Panel" and the "Electronic Attitude Display Indicator", respectively.

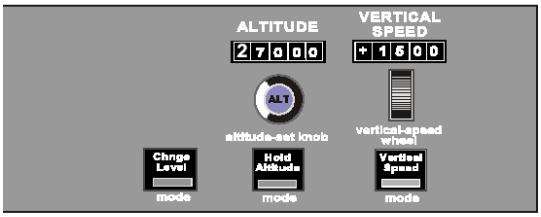

Figure 9(a). Guidance and Control Panel

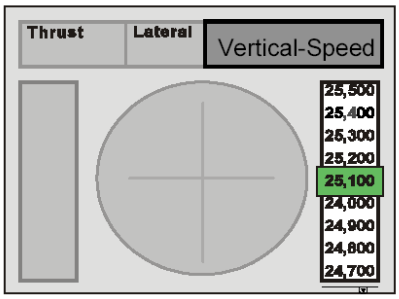

Figure 9(b). Electronic Attitude Display Indicator

Figure 9. Flight control interface.

Three primary vertical flight modes of the aircraft are of interest to us: the "Hold Altitude" mode, in which a specified altitude value is maintained constant, the climb/descend "Vertical Speed" mode, and the “Change Level” mode. These three modes can be engaged by pressing the respective buttons on the "Guidance Control Panel" (Figure 9(a)). In the top portion of the control panel are two windows, one indicating the altitude set by the pilot (set altitude), and the other indicating the vertical-speed setting. The pilot can change the altitude setting by rotating the altitude knob, and change the vertical speed by sliding the vertical-speed wheel up or down.

Figure 9(b) is a schematic illustration of the Electronic Attitude Display Indicator. It includes the “Flight Mode Annunciator” which indicates (in the top portion of the display) the current modes of the aircraft. In Figure 9(b) the current vertical mode, displayed in the right-most window, is “Vertical Speed” (the "THRUST" and "LATERAL" modes of the aircraft are beyond the scope of this paper). The altitude tape, which provides the pilot with an indication of the current altitude of the aircraft, is displayed in the right side of the display. By viewing both the “Guidance Control Panel" and the "Electronic Attitude Display Indicator”, the pilot has knowledge of the set altitude, the vertical speed setting, the active mode, and the current aircraft altitude.

Machine Model

Figure 10 is a fragment of the state transition diagram describing the transitions among the vertical-flight modes of the autopilot under consideration. The “Vertical Speed” (V/S) mode may be constrained by a target altitude (“V/S to altitude setting”), in which case a target altitude is armed for capture, or it may be unconstrained (“V/S unconstrained”), in which case no target altitude is armed for capture. In the "Change Level" mode the target altitude is always armed for capture. Also shown

are the transitory mode “Capture altitude setting” and the two “Hold altitude” modes. The transitions among the various modes are depicted as the labeled arrows to be explained in detail below.

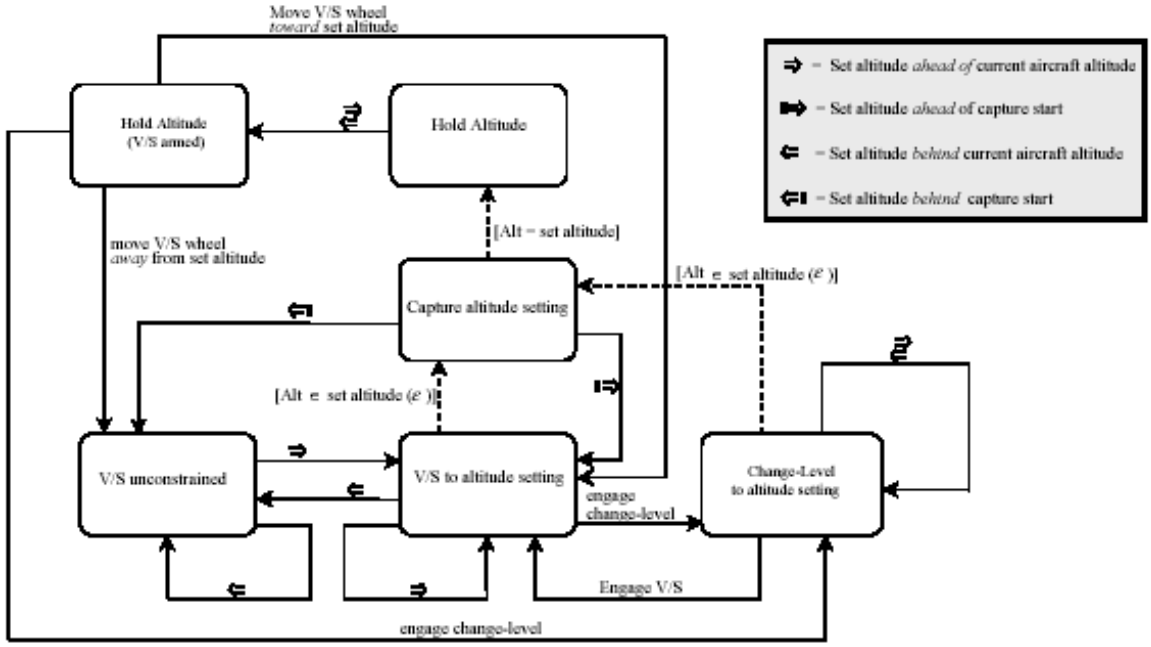

Figure 10. Machine model of vertical modes.

Each of the modes, represented by a rounded rectangle, can be thought of as a distinct aircraft activity, which is fully defined in the autopilot and has an associated set of parameters. For example, the “V/S to altitude setting” represents a climb or descent activity, and is parameterized by the value of the vertical speed setting (positive for a climb and negative for a descent) and the target-altitude setting. Similarly, the “V/S unconstrained” is only parameterized by the value of the vertical speed setting. The "Change Level" mode is fully specified by the target-altitude setting. The two “Hold altitude” activities are parameterized by the altitude setting. Finally, the “Capture altitude setting” is also parameterized by the altitude setting.

In this system some of the modes have associated dynamics, and some of the transitions among them are triggered by the internal dynamics of the system. Thus, in the "V/S to altitude setting" activity, the vertical speed parameter (which is not explicitly exhibited in Figure 10) determines the

rate-of-change in altitude. When the altitude reaches a value that satisfies the condition [Alt ∈ set altitude (ε)], an automatic transition to the mode "Capture altitude setting" takes place. This condition [Alt ∈ set altitude (ε)] is triggered by the equation

'altitude_error' + ['altitude_error_rate' * 'abs_val ('altitude_error_rate')] / (2 * Nz * g) = 0

becoming true, (where Nz is the normal acceleration during the capture phase).

The transition from "Change-Level to altitude setting" to "Capture altitude setting" is triggered by a similar condition. In the "Capture altitude setting" mode, the level-off maneuver to the target altitude (as set in the "Guidance Control Panel") is then executed. When the condition [Alt = set altitude] is met, an automatic transition to "Hold Altitude" is triggered by the autopilot. Automatic transitions are represented in the Figure by dashed lines. (Most of dynamical details of the autopilot are suppressed in the graphical description).

The pilot manually engages several mode transitions. These include the transition from "Change-Level" to "V/S to altitude setting" and its reverse transition, as well as the transition from "Hold Altitude (V/S armed)" to "Change-Level."

The remaining mode transitions are triggered indirectly by the pilot through the change of a parameter. These parameter changes sometimes trigger subtle mode transitions that cause significant changes in aircraft behavior and are a potential source for confusion and hazard. Analysis of such subtle transitions is the focus of the ensuing discussion.

The pilot can change the altitude setting at any time. There is a qualitative difference in the autopilot's behavior depending on whether the altitude is changed to a value ahead or behind a specific critical altitude (sometimes this is the current altitude of the aircraft and sometimes another value, as explained below). Here by *ahead* we mean in a temporal sense, that is, "higher than" the critical altitude when climbing, and "lower than" the critical altitude when descending. (Similarly, *behind* means "lower than" when climbing, and "higher than" when descending). These altitude setting

changes are shown in Figure 10 by the transition labels "=>" for ahead and "<=" for behind. Thus, when the autopilot is in the activity "V/S to altitude setting," changing the altitude setting to a value behind the current aircraft altitude triggers a transition to "V/S unconstrained." In this activity, the aircraft continues the current climb/descent without any effective altitude constraint. The reverse transition is triggered in the "V/S unconstrained" activity, when the altitude value is set ahead of the current altitude.

In the "Hold Altitude (V/S armed)" activity, moving the vertical speed wheel toward the set altitude results in a transition to "V/S to altitude setting," while moving the speed wheel away from the set altitude results in a transition to "V/S unconstrained." Of particular interest are the consequences of changing the altitude value while in the "Capture altitude setting" activity. Here the critical altitude is not the current aircraft altitude, but rather the altitude when the autopilot transitions into the "Capture altitude setting" mode (called henceforth the *capture-start* altitude) as can be seen in Figure 11.

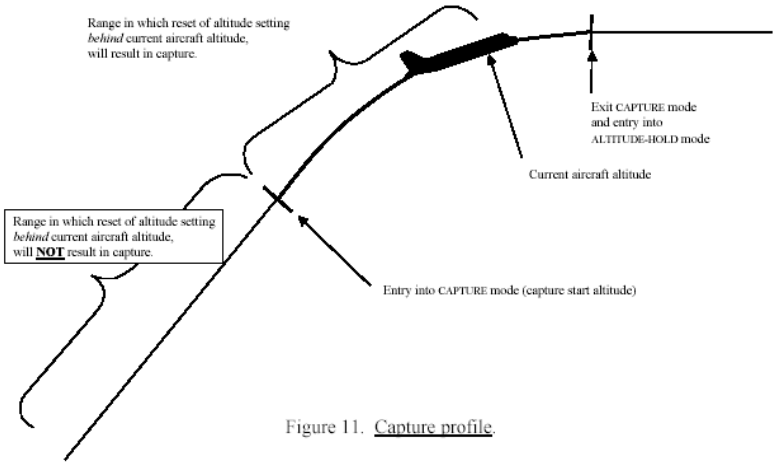

Figure 11. Capture profile.

Thus, in the "capture altitude setting" mode, changing the altitude to a value behind the *capture-start*, triggers a transition to "V/S unconstrained." On the other hand, changing the setting to a value ahead of capture-start, triggers a transition to "V/S to altitude setting", and the new altitude becomes the target altitude. The main point is that by resetting the altitude to a value ahead of capture-start, the

capture condition is retained, while resetting to a value behind capture-start results in an unconstrained climb or descent.

User-Model

Most of the verbal statements, in the aircraft-manual describing the behavior of this autopilot, have the following form: “when the autopilot is in mode X and button ‘k’ is pushed, the autopilot engages in mode Y”. It is therefore possible to collect these fragmented statements into a whole using a state-machine representation such as the one described in Figure 12. This will be our user model, and we shall give a brief explanation how it has been obtained from the training manual.

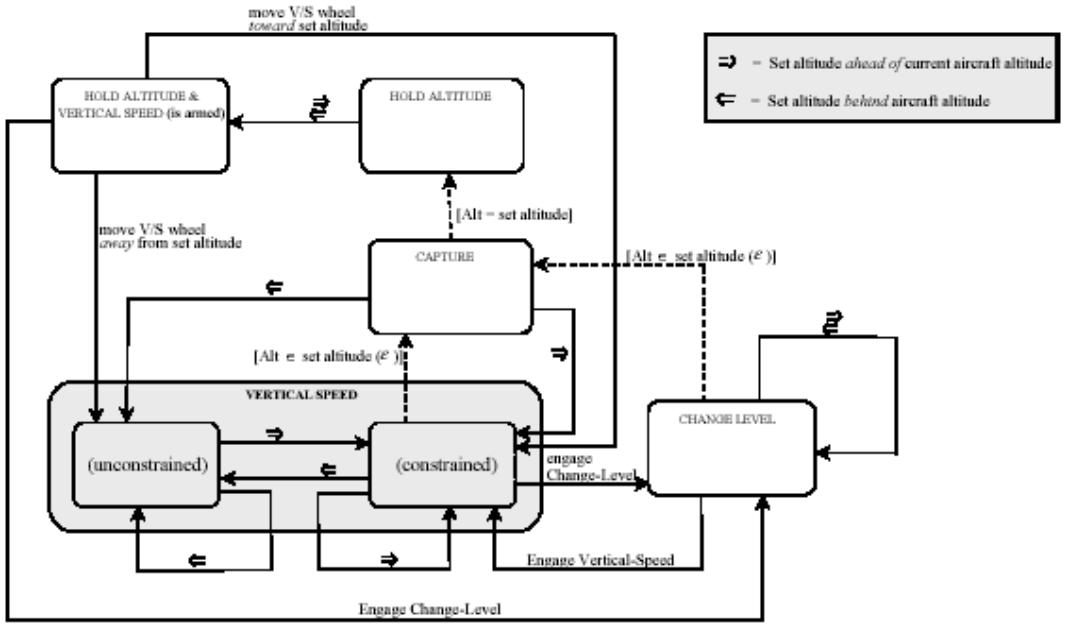

Figure 12. User model of vertical modes.

The vertical flight-modes that appear explicitly in the display are “Hold altitude”, “Hold altitude (Vertical Speed armed)”, “Capture”, “Change level”, and “Vertical Speed”. As before, let us focus on the aircraft's responses to changes of the altitude setting in the various modes. The following responses are described in the training manual: If the altitude is reset to a new value while in the "Vertical Speed" mode, the autopilot stays in the “Vertical Speed” mode. When the autopilot is in the “Capture”

mode, it responds to a change in the altitude setting by reverting back to the "Vertical Speed" mode. When the autopilot is in the “Hold altitude” mode, the autopilot responds to a change in the altitude setting by arming “Vertical Speed”, i.e., by moving to the mode “Hold altitude (Vertical Speed armed)”. From this latter mode, the autopilot moves to the “Vertical Speed” mode when the pilot changes the Vertical speed wheel to a new (non-zero) value. The transition “Engage change level” and “Engage Vertical Speed” are triggered by the pilot, while the transitions to “Capture” and to “Altitude hold” are automatic (represented by dashed lines).

In the display there is no indication of, and in the training material there is no explicit discussion on, whether the “Vertical Speed” mode will result in capture or not. Nevertheless, it is commonly accepted that the pilot has the ability to distinguish between these two Vertical-Speed sub-activities, based on whether the current vertical speed value is toward or away from the new altitude setting. Hence, we incorporated in the user-model these two sub-activities in dotted lines. Included are also (in dotted lines) the associated transitions between and into these sub-activities. Noteworthy, in particular, are the two transitions out of Capture mode that enter the vertical speed sub-activities according to the distinction made above.

Formal verification

We turn now to a formal verification of the adequacy of the user-model and interface of the autopilot. The correspondence between the autopilot events and the user-model (and interface) events is given in Table 1.

Σ_M	Σ_{usr}
Move V/S wheel *toward* set altitude	Move V/S wheel *toward* set altitude
move V/S wheel *away* from set altitude	move V/S wheel *away* from set altitude
Engage Vertical-Speed	Engage Vertical-Speed
Engage Change Level	Engage Change Level
⇄	⇄
[Alt = set altitude]	[Alt = set altitude]
[Alt ∈ set altitude (ϵ)]	[Alt ∈ set altitude (ϵ)]
⇒	⇒
⇨ (& ⇒)	
⇨ (& ⇐)	⇐
⇦	

Table 1. Correspondence between machine events and user-model events.

The first seven events in Table 1 are observed events. They are the same in the machine model and user model. The remaining events are masked events that consist of two groups, each comprising two events. Each of these groups is mapped to a single event in the user-model as seen in the Table. (There are no unobserved events in this example.)

The formal task specification consists of the requirement that at each instant and under all operating conditions, the pilot be able to deduce (unambiguously) whether or not the set altitude will eventually be captured. In analogy with the example of the previous section, where the specification consisted of the partition of the state-set into legal and illegal states, the current specification calls for the partition of the state-set into "capture" states (i.e., states where eventual automatic altitude capture is insured) and "non-capture" states. To verify whether the user-model and interface are correct for this specification, we turn to the procedure described in the previous section. Specifically, we construct the composite model by forming the masked synchronous product of the autopilot model of Figure 10 and the user-model of Figure 12.

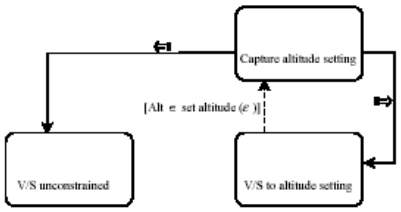

Figure 13(a). Machine model

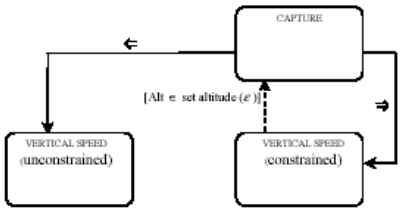

Figure 13(b). User model

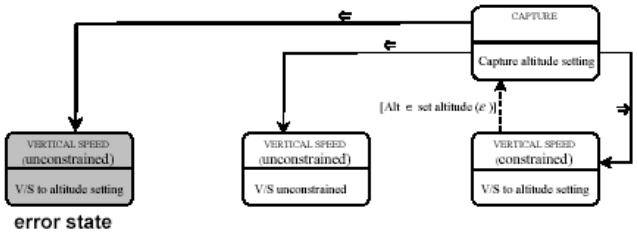

Figure 13(c). Composite model (from the user's point of view)

Figure 13. Correctness verification of vertical flight modes.

We focus on the transitions out of the capture mode as shown in Figure 13. The composite model of Figure 13(a) and Figure 13(b) is depicted in Figure 13(c). To understand how this composite model is obtained, let us refer again to Table 1. Notice that the transition ||=> ("set altitude ahead of capture start altitude") consists of two cases: (1) ||=> (&=>), namely, "set altitude ahead of capture start & ahead of current aircraft altitude", and (2) ||=> (& <=), namely, "set altitude ahead of capture start & behind current aircraft altitude". As can be seen in the table, the first case is masked in the user-model to => which means, "set altitude ahead of aircraft altitude", while the second case is masked to <=, namely "set altitude behind current aircraft altitude".

Therefore, the machine-model transition ||=> of Figure 13(a) is masked into either <= or into => depending on whether the (re)set altitude is behind or is ahead the current aircraft altitude. Consequently, when the aircraft is in capture mode and the altitude is reset to behind the 'current aircraft altitude', two different states may be entered depending on whether the altitude is reset behind or ahead of the 'capture start altitude'. In the latter case, the user-model enters the activity "Vertical

Speed (unconstrained)", which is *not* a capture-state, while the machine-model enters the activity "V/S to altitude setting", which is indeed a capture-state. This is a contradiction!

The composite-model of Figure 13(c) exhibits an error state and we must conclude that the user-model is incorrect for the task. From the pilot's perspective the autopilot behavior is very confusing – resetting the altitude to behind the current aircraft altitude sometimes leads to capture and sometimes it does not. This can be seen in Figure 13(c), where the same user-initiated event (<=), may lead to two different outcomes. Finally, it is worthwhile noting that this problem cannot be rectified by, simply, updating the user-manual and by providing additional training to flight crews. For the pilot to determine whether resetting the altitude will lead to capture or not, he or she needs to recall the capture start altitude. However, this value is not provided anywhere on the interface (see Figure 9(b)). A comprehensive solution to the problem will have to include modifications to the interface.

OPERATIONAL INCIDENTS

Following the completion of the analysis, it became interesting to check whether the observed design deficiency contributed to operational mishaps in actual practice. To this end, a search on NASA's Aviation Safety Reporting System (ASRS) was performed. (Following an incident, pilots can submit a detailed report to the ASRS, and are provided with certain assurances of anonymity and waiver of legal action against them (by the US Federal Aviation Administration, see Advisory Circular 00-46D)). The search revealed several incident reports, involving the specific autopilot described above, which referenced changing the altitude setting during capture. Below is one (slightly edited) excerpt. We shall first present the pilot's report and then our interpretation of the incident:

> On climb to 27,000 feet and leaving 26,500 feet. Memphis Center gave us a clearance to descend to 24,000 feet. The aircraft had gone to "Capture" mode when the first officer selected 24,000 feet on the GCP altitude setting. This disarmed the altitude capture and the aircraft continued to climb at approximately 300 feet-per-minute. There was no altitude warning and

> this "altitude bust" went unnoticed by myself and the first officer, due to the slight rate-of-climb. At 28,500, Memphis Center asked our altitude and I replied 28,500 and started an immediate descent to 24,000 feet.

In this incident, the first officer set a new altitude value (24,000) while the aircraft was climbing to 27,000 feet and in the "Capture" mode. As discussed earlier, changing the altitude setting to a value behind the capture-start altitude triggered a transition to "Vertical Speed" mode and to an unconstrained climb. The First Officer did not anticipate this aircraft response (and probably expected the aircraft to capture the new altitude setting).

SUMMARY AND IMPLICATIONS FOR DESIGN

In this paper we have investigated several aspects of the connection between the machine's behavior, the user's task, the required user-interface, and the user-model, for ensuring correct and unambiguous user-machine interaction. We have focused attention on situations where the machine's behavior and the user's model of the machine (as obtained, for example, from manufacturer's manuals, training materials, and the like) can be represented by formal models such as, in particular, state transition systems. The user's task is then specified in terms of a partition of the machine's states into distinct "specification-classes". (Examples of such specifications were given as partition into "legal"/"illegal" states, "armed"/"unarmed" states, differentiation between modes, etc.) In addition to the requirement that the user-model be free of blocking-states, the essential requirement for user model (and interface) correctness is that it be free of error-states, namely, that the user is able to *track* the specification classes while interacting with the machine.

As an application of our methodology, we have chosen to verify one element of pilot interaction with cockpit automation. We have demonstrated how ambiguities that exist with respect to the pilot's interaction with the vertical modes of a modern autopilot can be detected. The problem with resetting the altitude while in capture mode has not been understood previously and its occurrence has been

attributed to a variety of factors such as lack of situation awareness, training deficiencies, as well as autopilot malfunction. Yet the problem is directly attributable to the inadequacy of the interface and associated training material. Such design problems, which lie undetected in many aviation systems, confuse pilots and lead to faulty interactions.

Implications for design

We conclude with several comments regarding the implementation of the methodology and its implications for design of human machine interaction in complex automated systems.

In human machine interaction, two distinct aspects of the information provided to the user about the behavior of the machine play essential roles: (1) *what* information (nature and content) is provided to the user, and (2) *how* this information is presented and perceived. It is clear that both aspects must be addressed for a correct and efficient interaction. While the second aspect has received considerable attention in the literature, the first aspect seems to have been largely taken for granted.

The present paper focuses exclusively on the first aspect because of its acuteness in highly automated systems that are supervised by humans. We re-emphasize that to verify correctness of human-automation interaction, a formal model describing the behavior of the machine and a user-model must be available.

User-models represent information, which according to the designers' understanding, the user must know in order to operate the machine correctly. Our proposed methodology is aimed at a rigorous and systematic verification of this aspect of the designer's understanding of user-requirements.

In many practical cases, user-models evolve as the design changes and matures. They are constantly modified and updated. Conventional approaches for evaluation (and certification) of interfaces are based on extensive walk-throughs, testing, and simulations. These methods are time consuming, expensive, incomplete and therefore not fully reliable. Our proposed methodology also

provides a tool for error detection and progressive evaluation of the correspondence between the user-model, the interface and the machine.

The models addressed in this paper are purely of the Discrete Event type and the proposed methodology is presented within this framework. Many practical systems have also continuous and time dependent behaviors and cannot be fully modeled within the state transition framework discussed here. While the proposed approach is indeed general, its adaptation to timed and hybrid systems remains an open challenge for future research. Finally, while the paper discusses only the topic of verification, the approach and methodology has already been extended toward the development of a formal procedure for automatic generation of interfaces and user models that are succinct and correct (Heymann and Degani, 2002).

ACKNOWLEDGMENTS

This work was conducted as part of NASA's base research and technology effort, human-automation theory sub-element. The second author was supported by Grant NCC 2-798 from the NASA Ames Research Center to the San Jose State University. We gratefully acknowledge the support of one avionics firm and its staff for providing extensive simulator-time and engineering resources necessary to build and validate the models described in this paper. Special thanks are also due to Captain David Austin for flying the flight simulator and providing helpful insights. Kevin Jordan and Michael Shafto provided support to this project. The authors thank Rowena Morrison, Lance Sherry, Alex Kirlik, Victor Lebacqz, George Meyer, Alisa Baker, and Barry Crane for encouragement and helpful insights.

REFERENCES

Abbott, K., Slotte, S. M., and Stimson, D. K. (1996). *The interface between flightcrews and modern flight deck systems.* Washington, DC: Federal Aviation Administration.

Aviation Week and Space Technology. (1995). Automated cockpits: who's in charge? Part 1 and 2. 142(5 and 6),

Billings, C. E. (1996). *Aviation automation: The search for a human centered approach.* Hillsdale, NJ: Erlbaum.

Degani, A., Shafto, M., and Kirlik, A. (1999). Modes in Human-Machine Systems: Constructs, representation, and classification. *International Journal of Aviation Psychology, 9*(2), 125-138.

Degani, A. and Heymann, M., Meyer, G., and Shafto, M. (2000). *Some Formal Aspects of Human-Automation Interaction.* NASA Technical Memorandum 2000-209600. Moffett Field, CA: NASA Ames Research Center (can be downloaded in PDF format from http://ic.arc.nasa.gov/publications/number.html).

Heymann M., and Degani A. (2002). *On abstractions and simplifications in the design of human-automation interfaces.* NASA Technical Memorandum 2002-211397. Moffett Field, CA: NASA Ames Research Center (can be downloaded in PDF format from http://ic.arc.nasa.gov/publications/number.html).

Indian Court of Inquiry. (1992). Report on accident to Indian Airlines Airbus A-320 aircraft VT-EPN at Bangalore on 14th February 1990. Indian Government.

Javaux, D., and De Keyser, V. (1998). The Cognitive Complexity of Pilot-Mode Interaction: A Possible Explanation of Sarter and Woods' Classical Result. In G. Boy, C. Graeber and J. Robert (Ed.), *Proceeding of the International Conference on Human-Computer Interaction in Aeronautics Conference* (pp. 49-54). Montreal, Quebec: Ecole Polytechnique de Montreal.

Mellor, P. (1994). CAD: Computer aided disasters. *High Integrity Systems, 1*(2), 101-156.

Norman, D. A. (1990). The 'problem' with automation: inappropriate feedback and interaction, not 'over-automation.' *Phil. Trans. Research Society London, B 327*, 585-593.

Parasuraman, R., Sheridan, T.B., and Wickens, C.D. (2000). A model for the types and levels of human interaction with automation. *IEEE Transaction on Systems, Man, and Cybernetics – Part A: Systems and Humans, 30*(3), 286-297.

Rouse W. B. (1977). Applications of control theory in human factors-II [special issue]. *Human Factors, 19*(5).

Sarter, N. B., and Woods, D. D. (1995). How in the world did I ever get into that mode? Mode error and awareness in supervisory control. *Human Factors, 37*(1), 5-19.

Vicente, K. J. (1999). *Cognitive work analysis*. Mahwah, NJ: Lawrence Earlbaum.

Wickens, C. D. (1992). *Engineering psychology and human performance*. New York: Harper-Collins.

Wiener, E. L. (1989). *The human factors of advanced technology ("glass cockpit") transport aircraft* (NASA Contractor Report 177528). Moffett Field, CA: NASA Ames Research Center.

Wiener, E. L. (1993). Crew coordination and training in the advanced-technology cockpit. In E. L. Wiener, B. G. Kanki, and R. L. Helmreich (Eds.), *Cockpit resource management*. San Diego: Academic Press.

Wiener, E. L., Chute, R. D., and Moses, J. H. (1999). *Transition to glass: pilot training for high-technology transport aircraft* (NASA Contractor Report 208784). Moffett Field, CA: NASA Ames Research Center.

Woods, D., Sarter, N., and Billings, C. (1997). Automation surprises. In G. Salvendy (Ed.), *Handbook of human factors and ergonomics* (pp. 1926-1943). New York: John Wiley.

ASAF DEGANI and MICHAEL HEYMANN (*Formal verification of human-automation interaction*)

ASAF DEGANI received a M.S degree in Ergonomics from the University of Miami and a Ph.D. in Industrial and Systems Engineering from Georgia Institute of Technology, Atlanta. He holds a private pilot license and a maritime Captain ticket (international).

Since 1989 he has worked as a research scientist at the Human Factors Division at NASA Ames. Primary research topics are in human interaction with automation, modeling, design of procedures and decision aids, cockpit displays, and aviation safety.

MICHAEL HEYMANN received the B.Sc. and M.Sc. degrees from the Technion, Haifa, Israel, in 1960 and 1962, respectively, and the Ph.D. degree from the University of Oklahoma, Norman, in 1965, all in Chemical Engineering.

For the past 30 years he has been with the Technion, Israel Institute of Technology, where he is currently Professor of Computer Science and Director of the Center for Intelligent Systems, holding the Carl Fechheimer Chair in Electrical Engineering. He has previously been with the department of Applied Mathematics of which he was Chairman and with the Department of Electrical Engineering. Since 1983 he has also been associated with NASA Ames Research Center.

His research covered topics in the areas of linear system theory, differential games, optimization and adaptive control. His current interests are primarily in the areas of discrete event systems, hybrid systems, the theory of concurrent processes and various formal aspects of human-machine interaction. He has been on the editorial boards of the SIAM Journal of Control and Optimization and Systems and Control Letters.